****

**Lab YP2**

**Using Switchable CPU Clock**



**MIPSfpga 2.0. Lab YP2 - Using switchable clock to observe the CPU cycle-by-cycle**

**1. Introduction**

This lab introduces switchable clock, a feature of MIPSfpga 2.0 that enables the whole new category of student labs. With this feature you can first run the processor with its usual multi megahertz frequency, then switch it to the frequence of a few clocks per second and observe how the processor works live. A typical usage is to connect to external LEDs the CPU signals that control cache evictions or pipeline forwarding and observe the LED patterns when running different sequence of code. This usage is demonstrated in *MIPSfpga 2.0. Lab YP5 - The first glance into caches* and *MIPSfpga 2.0. Lab YP6 - The first glance into pipelining*.

**2. The theory of operation**

When implemented in ASIC using 28 nm technology, MIPS microAptiv UP can run up to 500 MHz; when implemented using 65 nm - more than 300 MHz.

When MIPS microAptiv UP is synthesized for FPGA together with block memory in MIPSfpga system, the frequency is much lower - around 50-60 MHz, both for Xilinx and Altera. The best case for any FPGA is around 150 MHz, with the smallest amount of block memory on some FPGAs.

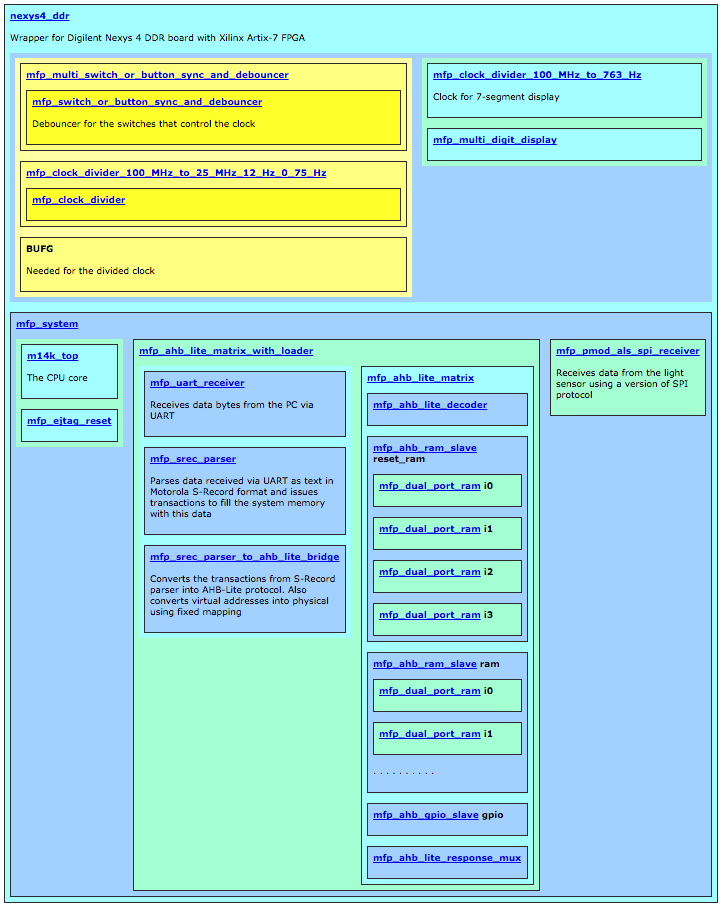
The introductory student boards tested for MIPSfpga all have clock generators able to generate a clock signal with the frequency of 50-100 MHz. This frequency can be increased or decreased using phase-locked loop (PLL). Unfortunately, PLL cannot be used to lower the frequency below approximately 100 KHz. In order to lower the frequency even further, other methods have to be used.

Altera has a special macro for such situation called *ALTCLKCTLR*, but it does not work for all cases. As a result, the switchable clock in MIPSfpga 2.0 is implemented using a combination of a counter and a global buffer (Xilinx macro *BUFG* and Altera macro *global*). The buffer is necessary, without it the synthesizer will not treat the output of the switchable clock module as a clock and the design will not work. It is also necessary to have timing constraints in the constraint file for Altera Quartus.

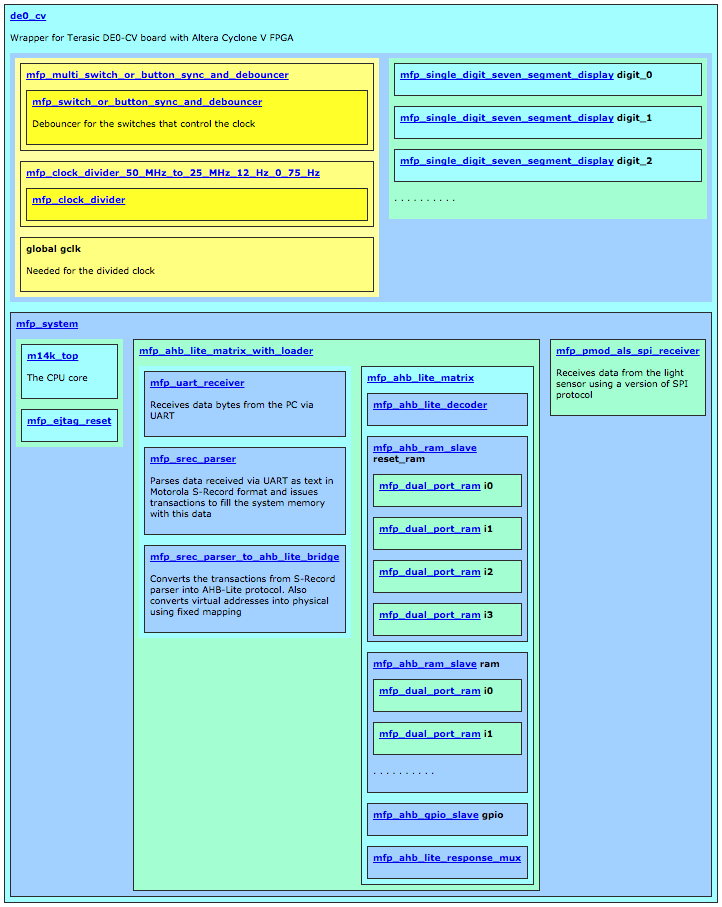
The frequency is controlled by two switches on the board. Switches also have to be debounced.

**Figure 1** show how the switchable clock module is instantiated in the module hierarchy for Digilent Nexys4 DDR board that carries Xilinx Artix-7 FPGA. **Figure 2** shows the same for Terasic DE0-CV board that carries Altera Cyclon V FPGA.

**Figure 1. MIPSfpga module hierarchy, including the switchable clock module, for [Digilent Nexys4 DDR board](http://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/) that carries Xilinx Artix-7 FPGA**

[](http://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/)

**Figure 2. MIPSfpga module hierarchy, including the switchable clock module, for [Terasic DE0-CV board](http://de0-cv.terasic.com) that carries Altera Cyclon V FPGA**

[](http://de0-cv.terasic.com/)

This lab also uses a special module to drive multi-digit seven-segment display on the board. A variant of such module for Digilent boards contains the code that handles dynamic displays and a variant for Terasic boards handles static displays.

**3. Lab steps**

This section outlines the sequence of steps, necessary to complete the lab. Almost all generic steps in this lab are the same as in *MIPSfpga 2.0 Lab YP1. Using MIPSfpga with Serial Loader Flow that does not require BusBlaster board and OpenOCD software*. Such generic steps are not described in this section. Only the steps different from *Lab YP1* are explained in details.

**3.1. Review the lab code modifications in *boards* and *system\_rtl* directories**

Search for *MFP\_USE\_SLOW\_CLOCK\_AND\_CLOCK\_MUX* symbol in *boards* and *system\_rtl* directories. Review the code fragments where that symbol occurs.

**3.1.1 System configuration**

Review and possibly modify the configuration parameters in the file *system\_rtl/mfp\_ahb\_lite\_matrix\_config.vh* as follows:

File *mfp\_ahb\_lite\_matrix\_config.vh*

//

// Configuration parameters

//

// `define MFP\_USE\_WORD\_MEMORY

// `define MFP\_INITIALIZE\_MEMORY\_FROM\_TXT\_FILE

`define MFP\_USE\_SLOW\_CLOCK\_AND\_CLOCK\_MUX

`define MFP\_USE\_UART\_PROGRAM\_LOADER

// `define MFP\_DEMO\_LIGHT\_SENSOR

// `define MFP\_DEMO\_INTERRUPTS

// `define MFP\_DEMO\_CACHE\_MISSES

// `define MFP\_DEMO\_PIPE\_BYPASS

**3.1.2 Review board wrapper file for Xilinx**

The board wrapper for Digilent Nexys4 DDR board with Xilinx Artix-7 FPGA instantiates three new modules:

1. A module that divides the clock - *mfp\_clock\_divider\_100\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz*
2. A module that debounces the switches that control the clock - *mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer*
3. A Xilinx-specific module *BUFG* that tells Vivado software that the generated signal should be treated as a clock by synthesis and routed like a clock during place-and-route

File *nexys4\_ddr.v*

`include "mfp\_ahb\_lite\_matrix\_config.vh"

module nexys4\_ddr

(

input CLK100MHZ,

input CPU\_RESETN,

. . . . . . . . . . . . . . . . . . . .

input [15:0] SW,

. . . . . . . . . . . . . . . . . . . .

);

wire clk;

`ifdef MFP\_USE\_SLOW\_CLOCK\_AND\_CLOCK\_MUX

wire muxed\_clk;

wire [1:0] sw\_db;

mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer

# (.WIDTH (2))

mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer

(

.clk ( CLK100MHZ ),

.sw\_in ( SW [1:0] ),

.sw\_out ( sw\_db )

);

mfp\_clock\_divider\_100\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

mfp\_clock\_divider\_100\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

(

.clki ( CLK100MHZ ),

.sel\_lo ( sw\_db [0] ),

.sel\_mid ( sw\_db [1] ),

.clko ( muxed\_clk )

);

BUFG BUFG\_slow\_clk (.O ( clk ), .I ( muxed\_clk ));

`else

clk\_wiz\_0 clk\_wiz\_0 (.clk\_in1 (CLK100MHZ), .clk\_out1 (clk));

`endif

. . . . . . . . . . . . . . . . . . . .

mfp\_system mfp\_system

(

.SI\_ClkIn ( clk ),

.SI\_Reset ( ~ CPU\_RESETN ),

. . . . . . . . . . . . . . . . . .

);

. . . . . . . . . . . . . . . . . . . .

endmodule

**3.1.3 Review board wrapper file and timing constraints for Altera**

The board wrapper for Terasic DE0-CV board with Altera Cyclone V FPGA instantiates three new modules:

1. A module that divides the clock - *mfp\_clock\_divider\_50\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz*
2. A module that synchronizes and debounces the switches that control the clock - *mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer. Without debouncing the control signals, the clock may not work reliably during the mode switching.*
3. An Altera-specific module *global* that tells Vivado software that the generated signal should be treated as a clock by synthesis and routed like a clock during place-and-route

File *de0\_cv.v*

`include "mfp\_ahb\_lite\_matrix\_config.vh"

module de0\_cv

(

input CLOCK2\_50,

input CLOCK3\_50,

inout CLOCK4\_50,

input CLOCK\_50,

input RESET\_N,

input [ 3:0] KEY,

input [ 9:0] SW,

. . . . . . . . . . . . . . . . . . . .

);

wire clk;

`ifdef MFP\_USE\_SLOW\_CLOCK\_AND\_CLOCK\_MUX

wire muxed\_clk;

wire [1:0] sw\_db;

mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer

# (.WIDTH (2))

mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer

(

.clk ( CLOCK\_50 ),

.sw\_in ( SW [1:0] ),

.sw\_out ( sw\_db )

);

mfp\_clock\_divider\_50\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

mfp\_clock\_divider\_50\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

(

.clki ( CLOCK\_50 ),

.sel\_lo ( sw\_db [0] ),

.sel\_mid ( sw\_db [1] ),

.clko ( muxed\_clk )

);

global gclk

(

.in ( muxed\_clk ),

.out ( clk )

);

`else

assign clk = CLOCK\_50;

`endif

. . . . . . . . . . . . . . . . . . . .

mfp\_system mfp\_system

(

.SI\_ClkIn ( clk ),

.SI\_Reset ( ~ RESET\_N ),

. . . . . . . . . . . . . . . . . .

);

. . . . . . . . . . . . . . . . . . . .

endmodule

It is also necessary to modify the timing constrains in the board-specific SDC file:

File *de0\_cv.sdc*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Create Clock

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

create\_clock -period 20 [get\_ports CLOCK2\_50]

create\_clock -period 20 [get\_ports CLOCK3\_50]

create\_clock -period 20 [get\_ports CLOCK\_50]

create\_clock -period 20 [get\_ports CLOCK4\_50]

create\_clock -period 40 [get\_nets mfp\_clock\_divider\_50\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz|clko]

create\_clock -period 20 [get\_nets de0\_cv|clk]

create\_clock -period 20 [get\_ports GPIO\_1[17]]

. . . . . . . . . . . . . . . . . . . . . .

**3.1.4 Review clock divider modules**

The clock divider modules use simple counter to generate a clock with the needed frequency:

File *mfp\_clock\_dividers.v*

module mfp\_clock\_divider

(

input clki,

input sel\_lo,

input sel\_mid,

output reg clko

);

// 50 MHz / 2 \*\* 26 = 0.75 Hz

parameter DIV\_POW\_FASTEST = 1;

parameter DIV\_POW\_SLOWEST = 26;

reg [DIV\_POW\_SLOWEST - 1: 0] counter;

always @ (posedge clki)

counter <= counter + 1'b1;

always @ (posedge clki)

clko <= sel\_lo ? counter [DIV\_POW\_SLOWEST - 1] :

sel\_mid ? counter [DIV\_POW\_SLOWEST - 5] :

counter [DIV\_POW\_FASTEST - 1] ;

endmodule

//--------------------------------------------------------------------

module mfp\_clock\_divider\_100\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

(

input clki,

input sel\_lo,

input sel\_mid,

output clko

);

mfp\_clock\_divider # (.DIV\_POW\_FASTEST (2), .DIV\_POW\_SLOWEST (27))

mfp\_clock\_divider (clki, sel\_lo, sel\_mid, clko);

endmodule

//--------------------------------------------------------------------

module mfp\_clock\_divider\_50\_MHz\_to\_25\_MHz\_12\_Hz\_0\_75\_Hz

(

input clki,

input sel\_lo,

input sel\_mid,

output clko

);

mfp\_clock\_divider # (.DIV\_POW\_FASTEST (1), .DIV\_POW\_SLOWEST (26))

mfp\_clock\_divider (clki, sel\_lo, sel\_mid, clko);

endmodule

. . . . . . . . . . . . . . . . . . . . . .

**3.1.5 Review the debouncing modules**

Synchronizing and debouncing the clock control signals is necessary for the clock to work reliably during the mode switching. Otherwise signal bouncing and a possibility of metastable state can introduce bugs in the work of the processor.

File *mfp\_switch\_and\_button\_debouncers.v*

module mfp\_switch\_or\_button\_sync\_and\_debouncer

# (

parameter DEPTH = 8

)

(

input clk,

input sw\_in,

output reg sw\_out

);

reg [ DEPTH - 1 : 0] cnt;

reg [ 2 : 0] sync;

wire sw\_in\_s;

assign sw\_in\_s = sync [2];

always @ (posedge clk)

sync <= { sync [1:0], sw\_in };

always @ (posedge clk)

if (sw\_out ^ sw\_in\_s)

cnt <= cnt + 1'b1;

else

cnt <= { DEPTH { 1'b0 } };

always @ (posedge clk)

if (cnt == { DEPTH { 1'b1 } })

sw\_out <= sw\_in\_s;

endmodule

//-------------------------------------------------------------------

module mfp\_multi\_switch\_or\_button\_sync\_and\_debouncer

# (

parameter WIDTH = 1, DEPTH = 8

)

(

input clk,

input [ WIDTH - 1 : 0 ] sw\_in,

output [ WIDTH - 1 : 0 ] sw\_out

);

genvar sw\_cnt;

generate

for (sw\_cnt = 0; sw\_cnt < WIDTH; sw\_cnt = sw\_cnt + 1)

begin : GEN\_DB\_B

mfp\_switch\_or\_button\_sync\_and\_debouncer

# (.DEPTH (8))

mfp\_switch\_or\_button\_sync\_and\_debouncer

(

.clk ( clk ),

.sw\_in ( sw\_in [sw\_cnt] ),

.sw\_out ( sw\_out [sw\_cnt] )

);

end

endgenerate

endmodule

**3.2. Connect the board to the computer**

For *Digilent* boards, such as *Nexys4*, *Nexys4 DDR* or *Basys3*, this step is obvious. For *Altera/Terasic* boards some additional steps required:

1. Connect USB-to-UART connector to FPGA board. Either *FT232RL* or *PL2303TA* that you can by from AliExpress or Radio Shack will do the job. *TX* output from the connector (green wire on *PL2303TA*) should go to pin 3 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano) and *GND* output (black wire on *PL2303TA*) should be connected to pin 6 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano). Please consult photo picture in *Lab YP1* to avoid short-circuit or other connection problems.
2. For *FT232RL* connector: make sure to set 3.3V/5V jumper on *FT232RL* part to 3.3V.
3. For the boards that require external power in addition to the power that comes from USB, connect the power supply. The boards that require the extra power supply include *Terasic DE2-115*.
4. Connect FPGA board to the computer using main connection cable provided by the board manufacturers. Make sure to put USB cable to the right jack when ambiguity exists (such as in *Terasic DE2-115* board).
5. Make sure to power the FPGA board (turn on the power switch) before connecting the UART cable from USB-to-UART connector to the computer. Failing to do so may result in electric damage to the board.
6. Connect USB-to-UART connector to FPGA board.

**3.3 Run the synthesis and configure the FPGA with the synthesized MIPSfpga system**

This step is identical to the synthesis step in *Lab YP1*

**3.4 Go to the lab directory and clean it up**

Under Windows:

cd programs\lab\_yp2

00\_clean\_all.bat

Under Linux:

cd programs/lab\_yp2

00\_clean\_all.sh

**3.5 Review the lab program code**

The lab program code is a simple counter. The value output to multi-digit seven-segment display is constructed in a way that shows the incrementing digits with both fast multi-MHz clock and slow ~1 Hz clock:

File *main.c*

#include "mfp\_memory\_mapped\_registers.h"

int main ()

{

int n;

for (n = 0;; n ++)

{

MFP\_RED\_LEDS = n;

MFP\_GREEN\_LEDS = n;

MFP\_7\_SEGMENT\_HEX = ((n >> 8) & 0xffffff00) | (n & 0xff);

}

return 0;

}

**3.6 Prepare the first software run**

Following the procedure described in *Lab YP1*, compile and link the program, generate Motorola S-Record file and upload this file into the memory of the synthesized MIPSfpga-based system on the board.

Under Windows:

1. cd programs\lab\_yp2
2. run 02\_compile\_and\_link.bat
3. run 08\_generate\_motorola\_s\_record\_file.bat
4. run 11\_check\_which\_com\_port\_is\_used.bat
5. edit 12\_upload\_to\_the\_board\_using\_uart.bat based on the result from the previous step - set the working port in “set a=” assignment.
6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. run 12\_upload\_to\_the\_board\_using\_uart.bat

Under Linux:

If uploading program to the board first time during the current Linux session, add the current user to *dialout* Linux group. Enter the *root* password when prompted:

sudo adduser $USER dialout

su - $USER

After that:

1. cd programs/lab\_yp2
2. run ./02\_compile\_and\_link.sh
3. run ./08\_generate\_motorola\_s\_record\_file.sh
4. run ./11\_check\_which\_com\_port\_is\_used.sh
5. edit ./12\_upload\_to\_the\_board\_using\_uart.sh based on the result from the previous step - set the working port in “set a=” assignment
6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. ./run 12\_upload\_to\_the\_board\_using\_uart.sh

**3.7 Run the software on the board**

Reset the processor. The reset buttons for each board are listed in the table below:

|  |  |
| --- | --- |
| **Board** | **Reset button** |
| Digilent Basys3 | Up |
| Digilent Nexys4 | Dedicated CPU Reset |
| Digilent Nexys4 DDR | Dedicated CPU Reset |
| Terasic DE0 | Button/Key 0 |
| Terasic DE0-CV | Dedicated reset button |
| Terasic DE0-Nano | Button/Key 0 |
| Terasic DE1 | Button/Key 0 |
| Terasic DE2-115 | Button/Key 0 |
| Terasic DE10-Lite | Button/Key 0 |

At this moment you should see the counter running and generating the output on seven-segment diplay on FPGA board. Turn the switch 0 on. This will switch the system clock from 25 MHz to 0.75 Hz, less than one beat per second. Can you see the sharp difference in counter output? If you don't, check the synthesis configuration.

Now try to turn switch 1 on and switch 0 off. What happened with the speed of counting? Now you are ready to use the switchable clock feature for the subsequent labs.